Joint Optimization of Response Time and Deployment Cost in Next-Gen IoT Applications

Juan Luis Herrera, Jaime Galán-Jiménez, José García-Alonso, *Member, IEEE*, Javier Berrocal, *Member, IEEE*, and Juan Manuel Murillo, *Member, IEEE*

***Abstract*—The irruption of the Internet of Things (IoT) has attracted the interest of both the industry and academia for their application in intensive domains, such as healthcare. The strict Quality of Service (QoS) requirements of the next generation of intensive IoT applications requires the QoS to be optimized considering the interplay of three key dimensions: computing, networking and application. This optimization requirement motivates the use of paradigms that provide virtualization, flexibility and programmability to IoT applications. In the computing dimension, paradigms such as edge or fog computing, Software-Defined Networks in the networking dimension, along with micro-services architectures for the application dimension, are suitable for QoS-strict IoT scenarios. In this work, we present a framework, named Next-gen IoT Optimization (NIoTO), that considers these three dimensions and their interplay to place micro-services and networking resources over an infrastructure, optimizing the deployment in terms of average response time and deployment cost. The evaluation of NIoTO in a healthcare case study reveals a response time speed-up of up to 5.11 and a reduction in cost of up to 9% with respect to other state-of-the-art techniques.**

***Index Terms*—Internet of Things, Software-Defined Networks, Computing in the Network, Edge Computing, Fog Computing, Quality of Service**

I. Introduction

The increase in the number of Internet-connected devices in recent years, especially caused by the irruption of the Internet of Things (IoT) paradigm, has led to an exponential growth of the amount of traffic flowing through the network. In particular, it is expected that the number of connected devices will be more than three times the global population by 2023. Indeed, Machine-To-Machine (M2M) connections will conform the half of such connected devices (reaching to 14.7 billion M2M connections) in that year [1].

Manuscript received January 00, 0000; revised January 00, 0000; accepted January 00, 0000. Date of publication January 00, 0000; date of current version January 00, 0000. This work has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), the 4IE+ Project (0499-4IE-PLUS-4-E) funded by the Interreg V-A España-Portugal (POCTEP) 2014-2020 program, by the Department of Economy, Science and Digital Agenda of the Government of Extremadura (GR21133, IB18030), by the Valhondo Calaff institution, and by the European Regional Development Fund. *(Corresponding author: Juan Luis Herrera.)*

The authors are with the Department of Computer Science and Communications Engineering, University of Extremadura, Spain (e-mail: jlherrerag@unex.es).

Digital Object Identifier 00.000/JIOT.0000.0000000

The computational IoT tasks and the related data processing may be computationally intensive, which can not be often accomplished by regular IoT devices with limited resources (memory, battery, etc.) [2]. To overcome this difficulty these tasks are usually offloaded to the cloud, where they can be executed without compromising available resources. However, a penalty derived from the latency imposed by the separation of end devices and the cloud must be paid.

While this approach can be easily used with elastic applications that do not have strict Quality of Service (QoS) requirements (e.g., voice assistants), next-generation IoT applications (e.g., autonomous driving, Internet of Medical Things (IoMT) applications) require greater bandwidth and ultra low-latency constraints that are not feasible with a pure cloud-based paradigm [3]. Edge and fog computing paradigms represent a suitable solution for these intensive computational tasks that must be executed under strict latency requirements [3], [4]. By moving services from the cloud to the edge of the network some benefits are obtained: i) a reduction in the required latency; and ii) the computational load of tasks is restrained at end devices, since they are offloaded to edge servers [2].

Nonetheless, the time required to execute a request in a distributed application, or response time, has two components: latency, and execution time. While the former depends on the networking fabric and the distance between end devices and those they offload their tasks to (e.g., cloud, edge nodes), the latter depends on the computational load of these tasks and the power of the devices running them. Therefore, three dimensions are involved in the QoS of IoT applications: the computing dimension, the networking dimension and the application dimension. Furthermore, the next generation of IoT applications require for virtualization, flexibility and programmability features in these three dimensions [5]. Emerging paradigms, such as edge and fog computing in the computing dimension, Software-Defined Networking (SDN) for the networking dimension, and micro-services architectures in the application dimension, can be enablers for the QoS-optimal deployment of next-gen IoT applications [2], [6]–[9]. However, the QoS experienced by applications running in such architectures depends on the performance provided by the set of computing and networking resources. Analogously, such performance depends on the way micro-services and the SDN controller are placed [4], [7]. Moreover, micro-services may be replicated to improve the QoS, at the cost of assessing the number of replicas that must be deployed for a given scenario. Therefore, in order to optimally deploy IoT applications through all three dimensions, the problem of placing micro-services as well as the SDN controller to maximize the experienced QoS must be carefully addressed.

In this paper, we provide a framework named Next-gen IoT Optimization (NIoTO) to optimize the placement of micro-services and networking resources to optimize the QoS considering the computing, networking and application dimensions. In particular, the proposed solution provides i) the optimal number of micro-service replicas to be placed in the infrastructure, ii) the optimal placement of micro-services to be run in the computing resources of the infrastructure, iii) the optimal placement of the SDN controller to reduce the required signalling cost, and iv) the optimal routing of the traffic flows generated by micro-services, as well as control traffic flows. To do so, the problem is modeled and formulated using Mixed-Integer Linear Programming (MILP). Based on a scenario where next-gen IoT applications are considered, the framework is evaluated considering two metrics: i) the minimization of the average response time, and ii) the minimization of the costs derived from the deployment of the architecture. Finally, we remark the benefits of jointly considering the application, computing and networking dimensions to handle today IoT applications with stringent QoS requirements. The results obtained in the evaluation show that the use of NIoTO as an optimization tool is feasible at design-time. Furthermore, these results show that NIoTO is able to achieve shorter response times and lower costs compared to other, state-of-the-art benchmarks. The authors of this paper presented an initial work, the DADO framework, in [10], focused on the optimization of response time in IoT applications through optimal micro-service deployment and SDN controller placement. NIoTO extends DADO by considering deployment cost along with response time, including the possibility of multi-objective optimization, as well as trading off cost and response time. Furthermore, unlike DADO, NIoTO has been evaluated under a scenario that allows for combined fog and cloud microservice deployment. The main contributions of this work are as follows:

- The proposal of the NIoTO framework as a contribution to the optimization of the response time and deployment cost in next-gen IoT applications. NIoTO is able to optimally assess the number of micro-service replicas to deploy, where to deploy each of them, where to place the SDN controller, and the optimal routing of both application and control traffic flows. Furthermore, NIoTO supports the optimization of both, response time and deployment cost, by finding the optimal trade-off between them.
- The description and development of a problem model that considers both objectives supported by NIoTO. This model provides a holistic consideration of the optimization process, taking into the account the effects of one of NIoTO's decisions in a dimension on the rest of the dimensions. This model is developed through mathematical programming techniques, and provided as a mathematical problem formulation.
- The experimental evaluation of NIoTO in a healthcare-based next-gen IoT application, focusing on the trade-off between the two objectives supported by NIoTO and the performance differences between NIoTO and related, state-of-the-art frameworks.

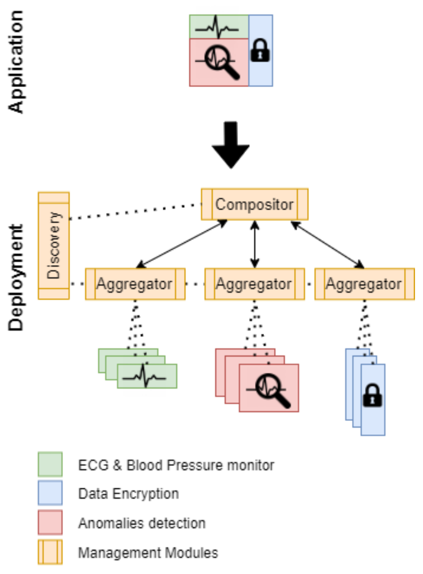
The remainder of the paper is organized as follows. Section II describes the architecture considered, remarking all the paradigms acting as enablers. Section III presents the proposed framework to optimize the application deployment, whose formulation is detailed in Section IV. Section V presents the performance evaluation of the framework over a healthcare scenario. Finally, Section VI concludes the paper.

II. Hierarchical multi-dimensional Architecture

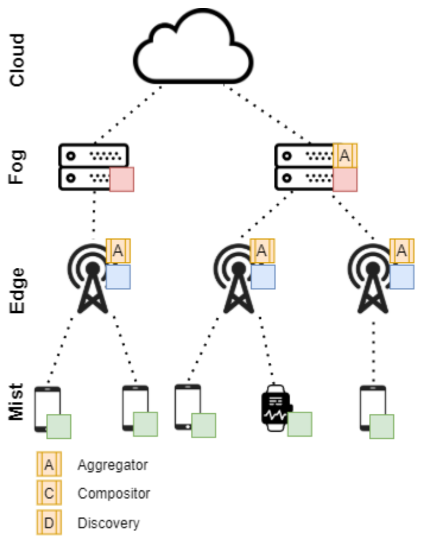
The development of QoS-stringent IoT applications requires the coordination of three highly related dimensions (Fig. 1): First, the application dimension, indicating how the software architecture of the application is modularized and coordinated. Second, the computing dimension defining the available computing resources and how these modules are deployed on them. Third, the networking dimension detailing which elements of the infrastructure will support the communications among modules. All these dimensions must be visualized in a holistic way. All of them are highly related and the configuration of one dimension impacts on the rest and, hence, the final QoS obtained.

The software architecture of next-gen IoT applications is usually based on the Service Oriented Computing paradigm (SOC) since they have to be massively distributed, interoperable and highly evolvable [8]. The Micro Service Architecture (MSA) pattern allows applications to be split into loosely coupled collaborating modules. These modules, usually called Bounded Contexts, APIs or, simply, services, contain one or more highly coupled micro-services that are usually deployed together [11]. MSAs are defined in contrast to monolithic architectures, in which the application may also be modularized, but all the modules require to be deployed together. Each of the micro-services in the MSA may be offered through different interfaces, which are normally comprised of a communication protocol and a data format. Some popular interfaces are web services (SOAP protocol and XML format) [12], gRPC services (HTTP/2 protocol and Protocol Buffers format) [13] or RESTful services (HTTP+JSON) [14]. While RESTful is currently the most popular interface and proposals for its use in IoMT exist [15], micro-services are not bound to a concrete interface.

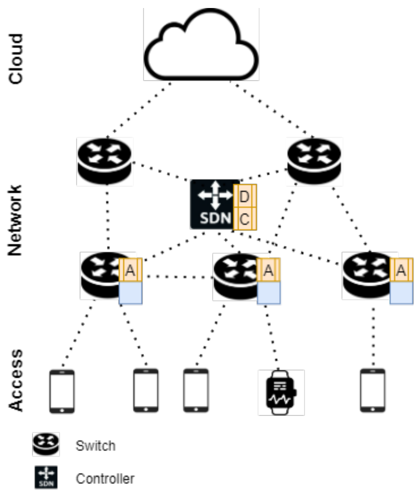
As a running example, which is depicted in Fig. 1, we base on an IoMT application to track the blood pressure of a patient and to detect anomalies in their electrocardiograms (ECG) by making use of data obtained through sensors. Each user is equipped with an IoT node that samples information for 15 seconds before sending it for further processing. The functionalities that are used in this case study are split into three services: ECG and blood pressure monitoring (green), data encryption (blue), and anomalies detection (red). Fig. 1a shows a high level architectural design of this application. This running example is based on the architecture proposed by [16].



(a) Application dimension.



(b) Computing dimension.



(c) Networking dimension.

Figure 1: Hierarchical IoMT application with the different dimensions affected to meet stringent QoS requirements.

In addition, their required data input, output and processing characteristics were obtained from [17].

These services can be deployed independently of one another, so that they can be deployed on the same or on different machines or servers, or even they can be replicated in order to balance the load and improve the QoS. Different virtualization and management technologies, such as Docker or Kubernetes, are used to reduce the effort and to automate this deployment. In addition, SOC makes use of some key technical foundations for the integration of the different services deployed [8], [18]: service discovery, for identifying the location of each service; service aggregation, for aggregating the responses; or service composition, coordinating the execution of complex functionalities by orchestrating several services.

The development of paradigms such as fog, edge or mist computing has allowed the development of intensive and QoS stringent IoT applications following a hierarchical architecture [2]. Services of MSA-based applications can be deployed closer to end-users in order to reduce network load and to improve response time [2]. The most resource-consuming services are still deployed on powerful nodes (such as cloud or, even, fog nodes) and the QoS stringent services, but less resource demanding, can be deployed on network nodes with computing capabilities –edge computing– or even on the Internet-connected devices themselves –mist computing– [2]. Therefore, a wide range of possibilities to deploy services must be evaluated, and in which nodes they are finally deployed is key to meet stringent QoS. Likewise, for complex functionalities, the deployed services may be coordinated, in order to achieve the functionalities and the workflows defined, using service discovery, aggregation and composition modules. The location of these management modules is also crucial in order to meet the required QoS.

Fig. 1b shows how the services defined on our running example are deployed on different layers of the computing hierarchical architecture depending on the required QoS and the available computing resources. For instance, the ECG and blood pressure monitoring service replicas (green service) is deployed on IoT devices, since this monitoring requires few computing resources. In addition, different aggregators (management modules) are also deployed on the upper layer (at the edge) to compute the responses of the IoT devices in order to reduce the data traffic, improving the response time. Likewise, data encryption (blue service) and anomaly detection (red service) both require some additional resources so that their replicas are deployed on more powerful nodes (edge and fog nodes, respectively). Therefore, defining an optimal computation distribution is highly dependent of the defined services and the available resources.

The deployment of an IoT application only taking into account the computing dimension does not always guarantee the optimal deployment in terms of QoS. In particular, the network configuration, such as the routing among the deployed services or the deployment of specific infrastructures, heavily impacts on the network latency and, hence, on the QoS. SDN networks allow the deployment of SDN controllers that leverage virtualization to make the network programmable based on the SOC principles [19]. Thus, Virtual Network Functions (VNFs) can be exploited for traffic engineering purposes in order to improve the network performance [20]. In this way, the SDN Controller Placement Problem (CPP) [7] identifies the deployment of SDN controllers in an optimal location to reduce the control latency and response time [21]. Fig. 1c shows that the SDN controller has been placed close to specific switches, in-between the Edge and Fog layers, in order to improve the average latency.

However, the network infrastructure has other capabilities that should also be reused. The application of virtualization and service oriented principles to the network infrastructure has led to a cloudification of both controllers and switches [19]. The Multi-Access Edge Computing (MEC) architecture embeds decentralized cloud capabilities in the network infrastructure at the network edge. This means that not only VNFs can be deployed on them, but also they can provision Virtual Computing Functions [19]. Therefore, IoT services and other management components (such as the service discovery, the aggregator or the compositor) can also be deployed in the network infrastructure. For instance, ETSI, the European Telecommunications Standard Institute, proposed the Management and Orchestration module (MANO) [22] to manage the different services and modules, their lifecycle, and to orchestrate them and chain different VNFs to support workflows. Therefore, the networking dimension is also able to perform some management tasks for the IoT applications. Fig. 1c shows that the service discovery and the service compositor modules have been deployed on the SDN controller in order to improve the QoS of the considered IoT application.

Therefore, a hierarchical multi-dimensional architecture boosts the achievement of stringent QoS requirements of next-gen IoT applications. Nevertheless, in order to achieve the optimal deployment and configuration, several dimensions must be jointly evaluated: the application dimension, the computing dimension and the networking dimension. In this paper, we base on an modularized IoT application using the principles of SOC and MSA [11], and propose a framework for the optimal placement of IoT services and SDN controllers in the hierarchical multi-dimensional architecture.

III. Next-gen IoT Optimization

Providing the best QoS requires a deployment that jointly evaluates the application, computing and networking dimensions to optimally place IoT services and SDN controllers. In this paper, we present the Next-gen IoT Optimization (NIoTO) framework, which takes all the three dimensions into account to find feasible deployments that meet the specific QoS requirements of IoT applications. Although NIoTO is an extensible framework that can support different types of QoS, in this paper, we focus on two specific requirements: response time and cost. These parameters are usually very important for IoT applications, as well as highly related —i.e, a lower response time usually requires a higher cost— and, therefore, a trade-off between them is difficult to achieve [23]. NIoTO takes as input the characteristics of the application, the network topology, the computing resources, and the QoS objectives to satisfy. These inputs are processed by the framework, which optimizes the QoS considering the three dimensions. Finally, results are reported as output, detailing the placement of the IoT services and SDN controllers to provide the optimal QoS according to the objectives.

A. Inputs

In order to know the optimal deployment design of an IoT application, different information is required for the three considered dimensions. These inputs are split into two types of information in order to improve its reusability and the extensibility of NIoTO to support other potential QoS requirements. These pieces of information are: *basic* information, and information that is *specific* to a QoS objective. The inputs required by NIoTO, split by their information type and their dimension, are depicted in a tree diagram in Fig. 2. These inputs are fed to NIoTO using open data interchange documents, such as XML or JSON, structured with a concrete schema, in which an arbitrary set of elements (e.g., micro-services, SDN switches, links, computing resources) can be defined. For each of these elements, the basic information, stored as its attributes, is mandatory, while QoS-specific information depends on the QoS to be optimized. Moreover, elements can be cross-referenced using their IDs, enhancing their reusability (e.g., a computing resource can be defined only once, and network links can refer to the definition through its ID). The use of non-proprietary data interchange formats eases the integration of NIoTO with other tools, as well as the creation of parsers to convert other formats to NIoTO inputs and vice-versa, making it easier to configure and use NIoTO. It is also possible to create support tools that assist the NIoTO user on the provision of the inputs (e.g., graphical user interfaces for NIoTO) using these data formats. Furthermore, each of the inputs are to be obtained at design-time. Some of these inputs can be obtained directly, while other inputs need to be estimated. For this estimation, the application is expected to be at the late design phase of development, in which the architectural decisions are already taken and low-level design has also been performed. Therefore, details such as the system's software architecture, the roles and functionalities of the micro-services, their complexity, or the planned network and computing infrastructure are known. Such knowledge is key, as it enables for the obtention of most parameters from the computing and networking dimensions directly. At this stage, the analysis of resource consumption, size, and performance in the application dimension, if performed with adequate techniques such as [24]–[26], can yield realistic estimates. The manner of obtention for each of the inputs, as well as its source, are detailed in Tab. I.

Basic information is always required independently from the QoS requirements to optimize. For the application dimension, the data required is the amount of RAM consumed by each service, the size of its inputs and outputs, and the requests that should be processed by each service (i.e., the workflows that are requested, and which device requests which workflow). For the computing dimension, the available computing resources (indicating their available RAM) and their location in the network topology must be provided as input. Finally, for the networking dimension, the network graph with the set of nodes and links, as well as link capacities, must be detailed. The three dimensions are used to specify the basic information in order to reduce the coupling. With this information, the feasibility of the deployment for the given IoT application is evaluated by checking if available resources are not exceeded.

QoS *specific* information are those inputs required to identify the optimal deployment for a particular type of QoS. Currently, NIoTO supports next QoS objectives: response time, deployment cost, and both of them (to find the best compromise between minimal response time and cost). Response time is calculated as the average response time for each request, which is the sum of the time needed to execute every service

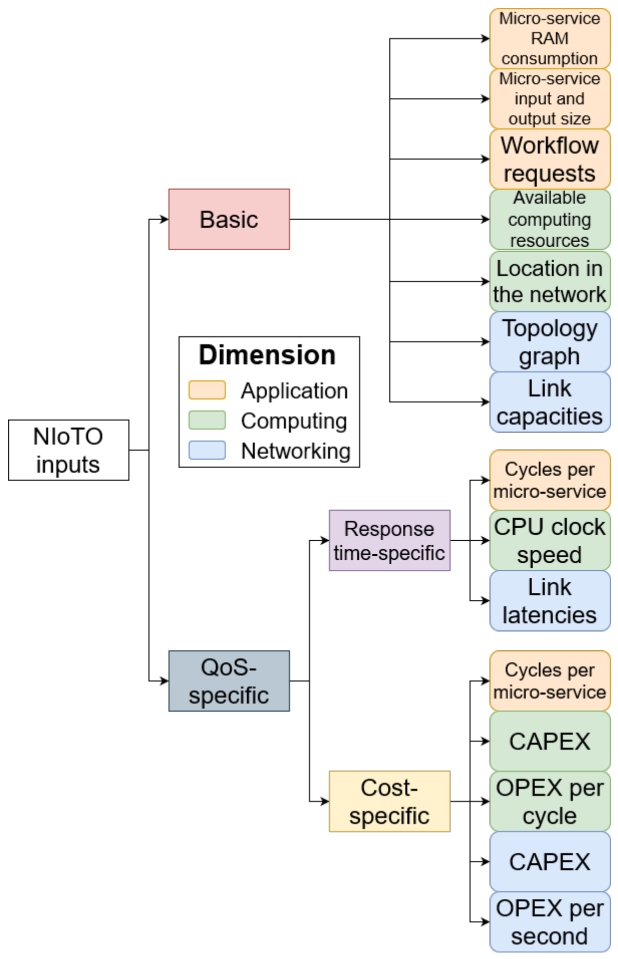


Figure 2: Inputs of the NIoTO framework per type of information and dimension.

in each request and the time needed to communicate to the devices where the services of such request are deployed. In order to evaluate this first objective, the inputs required for each of the three dimensions are: i) application dimension, the number of cycles required to execute each service; ii) computing dimension, the CPU clock speed of each computing resource; iii) networking dimension, the latency of each link. The cost objective, on the other hand, is assessed as the sum of the capital expenditures (CAPEX) (i.e., the cost of acquiring an asset) of each element used in the infrastructure plus the operational expenditures (OPEX) (i.e., the ongoing cost of maintaining an asset) derived from their use. Thus, for the application dimension, the number of cycles required by each service is required again. Instead, computing resources must include their CAPEX and OPEX per cycle. Network equipment must also report their CAPEX, as well as their OPEX per second, to optimize this objective.

To execute the NIoTO framework, to find feasible deployments, only basic information is required. If any given QoS objective must be optimized too, the QoS dependent information of said objectives has to be provided as well.

Table I: Kind of obtention and source of the input information for NIoTO at design-time.

Input information	Kind of obtention	Information source
Micro-service RAM consumption	Estimation	Memory complexity analysis of each micro-service
Micro-service input and output size	Estimation	Analysis of the expected input and output data type of each micro-service
Cycles per micro-service	Estimation	Time complexity analysis of each micro-service
Workflow requests	Estimation	Definition of the permitted workflows (use cases) for the application, estimation of the application's user base
Available computing resource	Direct	Planning of the computing devices to use, analysis of the datasheets and documentations of each planned device
Location of computing devices in the network	Direct	Network planning
CPU clock speed	Direct	Planning of the computing devices to use, analysis of the datasheets and documentations of each planned device
CAPEX of computing devices	Direct	Provided by the supplier
OPEX of computing devices	Direct (on demand)/Estimated (self-hosted)	Provided by the supplier (on demand)/planning of the computing devices to use, analysis of the datasheets and documentations of each planned device, calculation of derived cost (self-hosted)
Topology graph	Direct	Network planning
Link capacities	Direct	Planning of communication technologies, analysis of the capacity permitted by each planned technology
CAPEX of network equipment	Direct	Provided by the supplier
OPEX of network equipment	Estimated	Network planning, analysis of the datasheets and documentations of each planned networking device, calculation of derived cost

B. NIoTO framework

NIoTO takes all the inputs and provides the optimal deployment plan meeting the defined QoS objectives. To do so, NIoTO considers how services need to be executed in the computing resources, how executing these services in certain computing resources generates network traffic, and how the network manages such traffic. Following the example defined in Sec. II: given an example request for detecting anomalies in an ECG, if both services are deployed in the same machine, traffic will be sent back and forth between the IoMT device and said machine. However, if each service is deployed in a different machine, an additional traffic flow between both machines is required as well. The best deployment depends on (according to the QoS objective that has been set) the QoS the network is able to provide depending on the SDN controller placement and the QoS provided by each machine. NIoTO automatically selects the deployment that provides the best QoS considering all the three dimensions and their interplay. Nevertheless, a deployment plan that is optimal at some specific moment could be sub-optimal when the application context changes. Therefore, the IoT application deployment plan should be continuously evaluated and modified to ensure that the defined QoS is always met. There are two key phases with different time constraints to find the optimal deployment: design-time, and run-time. During the design phase, the application and infrastructure barely change their estimations, and there is no strict time limitation for obtaining the optimal deployment plan. Instead, during the execution, the infrastructure is dynamic and the monitored data often changes, and thus deployment plans are periodically needed. These periods cannot be very long, especially in cases in which a sudden spike in requests may require a deployment adaptation to maintain the QoS. The version of NIoTO presented in this paper is only suitable for design time optimization, due to the techniques used on its implementation. Nonetheless, the development and implementation of a version suitable for execution time, making use of other techniques that allow for faster optimizations, is one of our main future works.

To identify the optimal deployment plan at design-time, NIoTO makes use of MILP. This technique guarantees that the solution given as output meets all the constraints and is an optimal solution considering the given objective function. In order to support the three considered objectives (response time, cost and both), three different objective functions are defined. The MILP formulation of NIoTO tries to find an optimal deployment that places services and SDN controllers in a feasible way, respecting next constraints: i) RAM resources are not exceeded; ii) each traffic flow, regardless if it is control traffic or not, has a single source, a single destination and it is fully routed through a set of links (i.e., it is not divisible); and iii) link capacities are not exceeded. Then, the QoS objective function is applied so that the deployment is both feasible and optimal. In the case that both objectives are considered at the same time, i.e., response time and cost, NIoTO tries to find the optimal trade-off between them. While using or adding more resources to the architecture provides better response time, it is also costly, and vice-versa , thus NIoTO considers both to find the best compromise.

C. Outputs

NIoTO's output is a set of the placement decisions, i.e., a deployment plan for both IoT services and SDN controllers. The deployment plan follows certain patterns. For instance, each service replica needs to be deployed on the machine (or machines) it will be executed, the SDN controllers must be co-located with switches [7], each SDN switch is under the control of a single given SDN controller and traffic must be steered from the device requesting a service (or a set of them) to the machine where it is executed.

At design time, the deployment plan is a guide for the application's operation engineer, system administrator and network administrator. Information about which elements and how they should be used, as well as how to make an initial IoT application and SDN controller deployment are provided.

IV. Problem formulation

In this section, the MILP formulation that the NIoTO framework uses at design time is detailed. This version holistically optimizes the application, computing and networking dimensions. This formulation optimizes the number of microservice replicas required in the application dimension, where in the computing dimensions are each of the replicas deployed, as well as SDN controller placement and traffic routing in the networking dimension. This formulation is structured into four tightly coupled elements: parameters, decision variables, objective function, and constraints, that will be explained in the same order. Moreover, Table II provides a summary and quick reference of the notations used throughout the formulation.

A. Parameters

The parameters of a MILP formulation are its inputs: values that are provided to the formulation at run-time and stay fixed during the MILP solving process. Thus, the parameters of the formulation are the inputs of the NIoTO framework.

Let the deployment infrastructure be represented as a graph $G = \{V, L\}$. Let V be the set of vertices, which comprises both the computing and networking dimensions. The term *vertices* is used instead of *nodes* to avoid confusions with the term *fog nodes*. Let C be the set of computing devices in the infrastructure, and let S be the set of SDN switches, so that $V = C \cup S; C \cap S = \emptyset$. Furthermore, let L be the links that connect the infrastructure's vertices, i.e., $l_{ij} \in L; i, j \in V$.

The basic information of each computing device $c \in C$ comprises its amount of available RAM, r_c, measured in bytes. For the response time objective, QoS-specific information includes the device's CPU clock speed in Hz, P_c. On the other hand, deployment cost-specific information comprises the CAPEX of the device, $CAPEX_c$, and the device's OPEX per cycle, $OPEX_c^{\Omega}$.

For each SDN switch $s \in S$, its basic information is related to its position within the infrastructure, which is already represented in G. Thus, only cost-specific information is required: its CAPEX $CAPEX_s$, the CAPEX of an SDN controller placed in the switch $CAPEX_s^{CNT}$, its OPEX per second $OPEX_s$, and the analoguous OPEX of a controller $OPEX_s^{CNT}$. Response time QoS-specific information, such as control latency, depends on controller placement, and thus cannot be known a priori. Therefore, control latency is not an input, it is an internal calculation of the formulation instead. Finally, the size of the control packets used on the SDN

Table II: List of formulation notations.

Parameter	Meaning
G	Graph that represents the infrastructure
V	Set of vertices of G
C	Subset of V that are computing devices.
S	Subset of V that are SDN switches
L	Set of links of G
W	Set of workflows of the IoT application
r_c	RAM memory of the computing device c
P_c	CPU clock speed of the computing device c
$CAPEX_c$	CAPEX of the computing device c
$OPEX_c^{\Omega}$	OPEX per cycle of the computing device c
$CAPEX_s$	CAPEX of the SDN switch s
$OPEX_s$	OPEX of the SDN switch s
$CAPEX_s^{CNT}$	CAPEX of placing a controller on the SDN switch s
$OPEX_s^{CNT}$	OPEX of the controller placed on the SDN switch s
σ	Size of the SDN control packets
θ_{ij}	Capacity of the link l_{ij}
δ_{ij}	Latency of the link l_{ij}
$WS(w, c)$	Boolean function that indicates whether the workflow w is requested by the device c or not
I_m	Size of the input of the micro-service m
O_m	Size of the output of the micro-service m
Ω_m	CPU cycles of the micro-service m
ϵ_{OBJ}	Boolean that indicates whether the QoS objective OBJ is enabled or not
u_c	Boolean that indicates whether the computing device c is in use or not
u_s	Boolean that indicates whether the SDN switch s is in use or not
Decision variable	**Meaning**
$z_{cm_a}^w$	Boolean to determine if the micro-service m_a of the workflow w is deployed to the computing device c
$f_{ij}^{cwm_a}$	Boolean to determine if the traffic generated by the computing device c as a consequence of the micro-service m_a of the workflow w is routed through the link l_{ij}
x_s	Boolean to determine if a controller is placed on the SDN switch s
$y_{ss'}$	Boolean to determine if the SDN switch s is mapped to the controller placed on SDN switch s'
cf_{ij}^s	Boolean to determine if the control traffic generated by the SDN switch s is routed through the link l_{ij}

network, σ, is required as basic information. Nonetheless, σ does not depend on the SDN controller's deployment, but on the version of the OpenFlow protocol used [27].

The links that bind the infrastructure's vertices together have a limited available capacity θ_{ij}, which is part of their basic information. For the response time objective, each link's latency δ_{ij} must be known.

On the application dimension, NIoTO models execution as *functionalities*. A functionality, or *workflow*, is a request for the execution of a micro-service, or the execution of a set of micro-services in an ordered, pipelined manner. Following the example from Sec. II, a functionality may request for the execution of the ECG analyzer, the anomaly detector and the encryption service, so that the commented ECG outputted by the first micro-service serves as input to the anomaly detector, and the anomaly information is later encrypted for its safe storage. Thus, let W be the set of workflows requested for the application. Each workflow $w \in W$ is requested by a certain computing device, therefore, let $WS(w, c)$ be a binary function that evaluates to 1 if c requests workflow w and 0 otherwise. Moreover, let each workflow $w \in W$ be an ordered set of micro-services $w = \{m_1, m_2, ..., m_{|w|}\}$.

Each of these micro-services requires basic information, namely, the size of its input and output data, I_m and O_m respectively, and the amount of RAM it consumes, r_m. The QoS-specific information for response time also includes the number of CPU cycles that the micro-service requires to fully execute, Ω_m. Micro-services do not need any cost-specific information, as the costs derived from micro-services are a consequence of their deployment and execution, and thus, depend on the resources from the networking and computing dimensions used to execute them.

Finally, in order to enable and weigh the importance the QoS objectives, we add a parameter named ϵ_{OBJ}, which is a positive real number, or 0. ϵ_{OBJ} represents the weight of the objective OBJ in the optimization. It is important to note that the sum of all the values of ϵ_{OBJ} through all objectives must be exactly 1.

B. Decision variables

Similar to how parameters are the inputs of the MILP formulation, decision variables can be seen as its output: a set of values that the formulation must manipulate in order to find the optimal combination of values. Nonetheless, MILP is only able to manipulate integer decision variables, thus conditioning the information representation of the output. Concretely, NIoTO makes use of binary decision variables to represent information.

Firstly, the micro-service replication deployment must be modeled. To do so, binary variables are used: let $z_{cm_a}^w$ be a binary variable that takes a value of 1 if the micro-service m_a of the workflow w is deployed to the computing device c, and 0 otherwise. These variables automatically account for the number of replicas: if different workflows have the same micro-service deployed to the same computing device, they are sharing a single replica. Otherwise, multiple replicas exist. Secondly, the traffic flows generated by the communications between computing devices (i.e., to request the execution of the next micro-service in the workflow to another computing device) must also be modeled. Thus, let $f_{ij}^{cwm_a}$ be a binary variable that takes the value of 1 if the traffic flow generated by the computing device c as a consequence of the execution of the micro-service m_a of the workflow w is routed through the link l_{ij}, and 0 otherwise. With these decision variables, NIoTO is able to plan the replication of micro-services at the application dimension, the deployment of the replicas at the computing dimension, and the routing of the application's traffic through the networking dimension.

Nonetheless, NIoTO must also take the control of the networking dimension into account, and thus, place SDN controllers accordingly. To do so, let x_s be a binary variable, that is 1 if there is an SDN controller placed in the switch s and 0 otherwise. Nonetheless, if multiple controllers are placed, the mapping between SDN controllers and switches (i.e., which controller is on charge of which switches) must also be found. Hence, let $y_{ss'}$ be a binary variable, which takes the value of 1 if switch s is mapped to controller s' and 0 otherwise. Finally, NIoTO needs to account for the control traffic flows generated by the communications between switches and controllers to optimally route it. Let cf_{ij}^{s} be a binary variable that takes the value of 1 if the control traffic generated by switch s is routed through link l_{ij} and 0 otherwise.

C. Objective function

While MILP formulations manipulate the values of their decision variables to provide optimal outputs, they require a manner to calculate the optimality of the output. This manner is the objective function, which is a function of the decision variables. It is important to note that MILP only supports linear functions, i.e., it is strictly forbidden to multiply or non-linear operations over decision variables (e.g., multiplying two decision variables).

To better understand the objective function of NIoTO, we first define different calculations that will later be integrated. First, in order to calculate the execution time of a micro-service, the number of cycles of the micro-service need to be divided by the CPU clock speed of the computing device it is deployed to. Thus, if the micro-service m_a is deployed to the computing device c, then its execution time is $\frac{\Omega_{m_a}}{P_c}$. However, it is not possible to know *a priori* where each micro-service is executed. What can be known is that, given that m_a is part of workflow w, the variable $z_{cm_a}^{w}$ will have the value 1 if it deployed to c and 0 otherwise. Based on this knowledge, we can calculate the total execution time of a micro-service in a workflow by calculating the sum of all possible execution times in all computing devices, multiplied by $z_{cm_a}^{w}$:

$$EXEC_{m_a}^{w} = \sum_{c \in C} \frac{\Omega_{m_a}}{P_c} z_{cm_a}^{w}$$

Therefore, we define the execution time of a workflow as the sum of the execution times of its micro-services:

$$EXEC_w = \sum_{a=1}^{|w|} EXEC_{m_a}^{w}$$

Next, workflow latency needs to be calculated. In the case of a single flow that traverses a single link l_{ij}, latency is defined as the latency of the link, i.e., δ_{ij}. In a similar manner to the z variable of micro-services, the binary variable that contains the information of whether a flow traverses a link or not is $f_{ij}^{cwm_a}$. Nonetheless, there is an exception in the case of latency: if the flow reaches an SDN switch, i.e., if j is an SDN switch, we need to calculate the control latency of the switch, as it may need to communicate with the SDN controller. In this case, the binary variables for SDN control traffic flows is cf_{ij}^{s}. Thus, we can define the SDN control latency of a switch as:

$$CNTLAT_s = \sum_{l_{ij} \in L} cf_{ij}^{s} \delta_{ij}$$

Thus, with a known control latency, the latency of a traffic flow from a micro-service to another in the context of a workflow is is: i) its own latency if j is not an SDN switch, or ii) its own latency plus the control latency of j otherwise. To define it with more ease, let $SW(i)$ be a function which yields 1 if $i \in S$ and 0 otherwise. Formally, it is possible to denote this latency as:

$$LAT_{m_a}^{w} = \sum_{c \in C} \sum_{l_{ij} \in L} (f_{ij}^{cwm_a} \delta_{ij} + SW(j) CNTLAT_j)$$

Hence, to calculate the total latency of a workflow is the sum of the latencies of its micro-services:

$$LAT_w = \sum_{a=1}^{|w|} LAT_{m_a}^{w}$$

Thus, we define the average response time of the deployment as the average response times of all the workflows requested, each of them being the sum of the execution time and latency of the workflow:

$$RT = \frac{1}{|W|} \sum_{w \in W} EXEC_w + LAT_w$$

For the deployment cost objective, we need to split it in CAPEX and OPEX. In both cases, we assume that, since NIoTO operates at design time, any equipment that is not used will not be acquired, and hence, only the CAPEX of the used equipment should be considered. Thus, there is a need to know whether an element is in use or not. In the case of computing devices, we assume that one is used if it either requests at least one workflow, or runs at least one micro-service:

$$u_c = \max\left(\max_{w \in W, a \in [1,|w|]} (z_{cm_a}^{w}), \max_{w \in W} (WS(w,c))\right)$$

In the case of switches, a switch is used if it belongs to the route of at least one traffic flow, be it an application or a control flow:

$$u_s = \max\left(\max_{l_{is} \in L, c \in C, w \in W, a \in [1,|w|]} (f_{is}^{cwm_a}), \max_{l_{is} \in L, s' \in S} (cf_{is}^{s'})\right)$$

Thus, to obtain the CAPEX, we must multiply the CAPEX of each of the elements by 0 if they are unused, and by 1 if they are in use, and then sum the results. Formally:

$$\begin{aligned} CAPEX = &\sum_{c \in C} (CAPEX_c u_c) + \\ &\sum_{s \in S} (CAPEX_s u_s + CAPEX_s^{CNT} x_s) \end{aligned}$$

For the OPEX, we also need to account for the usage in cycles of the computing devices. Formally:

$$\begin{aligned} OPEX = &\sum_{c \in C} \left(\sum_{w \in W} \sum_{a=1}^{|w|} OPEX_c^{\Omega} \Omega_{m_a} z_{cm_a}^{w} \right) + \\ &\sum_{s \in S} (OPEX_s u_s + OPEX_s^{CNT} x_s) \end{aligned}$$

For the final objective function, we should take into account whether each of the QoS objectives is or is not enabled. Moreover, it is recommended to normalize each of the objectives so the final values on each side are within the same range (e.g., between 0 and 1), especially if both objectives are enabled. The final objective function is shown in Equation (1).

$$\min \epsilon_{RT} RT + \epsilon_{COST} (CAPEX + OPEX) \tag{1}$$

D. Constraints

With only parameters, decision variables and an objective function, an MILP formulation will find the combination of values for the variable that optimizes the objective function. However, this may lead to illegal situations in our problem: the limited capacities of computing devices and links could be surpassed, micro-services could have zero replicas, the SDN controller could be undeployed, switches may not be related to SDN controllers... In order to make the solution legal within the problem, we enforce these rules through constraints.

The constraints of NIoTO are as follows:

$$\sum_{c\in C} z^{w}_{cm_a} = 1 \forall w \in W, a \in [1, |w|] \tag{2}$$

$$\sum_{w\in W}\sum_{a=1}^{|w|} z^{w}_{cm_a} r_{ma} \le r_c \forall c \in C \tag{3}$$

$$\sum_{s'\in S} y_{ss'} = 1 \forall s \in S \tag{4}$$

$$y_{ss'} \le x_s \forall s, s' \in S \tag{5}$$

$$\sum_{j\in V} f^{cwm_1}_{ij} - f^{cwm_1}_{ji} = \begin{cases} 0 & \text{if } i \in S \\ WS(w,c)(1-z^{w}_{im_1}) & \text{if } i = c \\ -WS(w,c) z^{w}_{im_1} & \text{otherwise.} \end{cases} \tag{6}$$
$$\forall i \in V, c \in C, w \in W$$

$$-z^{w}_{cm_{a-1}} + z'^{iw}_{cm_a} \le 0 \tag{7}$$

$$-1 + z^{w}_{im_a} + z'^{iw}_{cm_a} \le 0 \tag{8}$$

$$z^{w}_{cm_{a-1}} + 1 - z^{w}_{im_a} - z'^{iw}_{cm_a} \le 1 \tag{9}$$

$$-z^{w}_{cm_{a-1}} + z''^{iw}_{cm_a} \le 0 \tag{10}$$

$$-z^{w}_{im_a} + z''^{iw}_{cm_a} \le 0 \tag{11}$$

$$z^{w}_{cm_{a-1}} + z^{w}_{im_a} - z''^{iw}_{cm_a} \le 1 \tag{12}$$

$$\sum_{j\in V} f^{cwm_a}_{ij} - f^{cwm_a}_{ji} = \begin{cases} 0 & \text{if } i \in S \\ z'^{iw}_{cm_a} & \text{if } i = c \\ -z''^{iw}_{cm_a} & \text{otherwise.} \end{cases} \tag{13}$$
$$\forall i \in V, c \in C, w \in W, a \in [2, |w|]$$

$$\sum_{j\in V} cf^{s}_{ij} - cf^{s}_{ji} = \begin{cases} 0 & \text{if } i \in C \\ 1 - y_{si} & \text{if } i = s \\ -y_{si} & \text{otherwise} \end{cases} \tag{14}$$
$$\forall i \in V, s \in S$$

$$\sum_{c\in C}\sum_{w\in W}[(\sum_{a=1}^{|w|} f^{cwm_a}_{ij} I_{m_a})] + \sum_{s\in S}[cf^{s}_{ij}\sigma] <= \theta_{ij} \tag{15}$$
$$\forall l_{ij} \in L$$

Equation (2) guarantees that each micro-service in a workflow is instantiated exactly once. Equation (3) enforces the RAM limit on the computing devices. Equation (4) makes sure each switch is controller by a single SDN controller, which must first be deployed as of Equation (5). Equation (6) guarantees that the traffic flow of starting a workflow has the workflow's requester as its source and the computing device that has deployed the first micro-service of the workflow as its destination, which is an special case of the general flow constraint from Equation (13). However, for micro-services beyond the first of a workflow, it is required to know if the machine that executed the previous micro-service is the same as the one hosting the current one, which would require variable multiplication. Since only linear constraints can be used in MILP, Equations (7-12) are used to *linearize* the problem by exploiting the properties of binary variables, which guarantee that $z'^{iw}_{cm_a} = z^{w}_{cm_a}(1 - z^{w}_{im_a})$ and $z''^{iw}_{cm_a} = z^{w}_{cm_{a-1}} z^{w}_{im_a}$. For control flow, Equation (14) has the same role. Finally, Equation (15) enforces link capacity.

By setting the parameters of the MILP formulation, an user can apply the NIoTO framework to hierarchical multi-dimensional architectures, making it possible to optimize the deployment of arbitrary next-gen IoT applications as long as they follow the NIoTO model.

V. Performance Evaluation

In this section, the possible benefits achieved when NIoTO is applied are evaluated: shorter response times and lower costs. These benefits are the result of NIoTO's consideration of the application, computing and networking dimensions by assessing the number of micro-service replicas, deploying the micro-services and placing SDN controllers, respectively. Moreover, the potential drawbacks of NIoTO are also analyzed. At first, the simulation environment is described. Then, three different sets of experiments are proposed. In the first one, the MILP solver is run over four topologies of different sizes with different optimization objectives to evaluate the trade-off between the deployment cost and the response time. The second analysis aims at evaluating the computational complexity of NIoTO by assessing the time required to optimize different scenarios, while the third one is devoted to comparing NIoTO's performance in both, response time and cost, with that of state-of-the-art benchmarks. This last analysis is carried out by comparing the response time and costs yielded by NIoTO with those obtained with the benchmarks.

A. Simulation Environment

In order to evaluate the performance of NIoTO framework, four different scenarios based on the example described in Sec. II have been considered. In each scenario, we vary the parameters of the three considered dimensions. In the application dimension, we vary the number of users (and thus, of workflow requests). In the computing dimension, we vary the number of fog nodes where such services can be deployed, as well as the number of gateway nodes connecting the network fabric to the cloud. The number of SDN nodes of the network is varied in the networking dimension. To evaluate the scalability of the proposed solution, values for previous parameters are increased in each scenario, deriving in the simulation set-up shown in Tab. III. The different network topologies leveraged for evaluation were generated using the Erdös-Rényi model for network generation [28], applying the parameters for the simulation set-up as reported by Tab. III. The specific model of each element in the scenario, as well as their CAPEX and OPEX (retrieved from [29]), are detailed in Tab. IV. The specific technical characteristics of these devices, such as their available RAM or CPU clock speed, were retrieved from the official specifications of each device. Thus, to retrieve this information, we refer the reader to the official datasheets and documentations of the computational resources. Regarding the type of wireless connection used by IoT devices to connect with the SDN network, both Wi-Fi and Bluetooth technologies are exploited by Arduino devices, while 6LoWPAN and ZigBee technologies are used by Texas Instruments ones. The access layer makes use of these wireless technologies, while the network core is connected through Gigabit Ethernet links. Link capacities were adjusted according to the capacity of the link's technology (e.g., Gigabit Ethernet links have 1 Gbps capacity, Wi-Fi links have 300 Mbps). To calculate the length of the links, the hospital in [30] is used as a reference for size. Thus, each of these links are, at most, 90 meters long, and their lengths were obtained from a normal distribution of mean $\mu = 45$ and standard deviation $\sigma = 18$. Their transport latencies were calculated based on these lengths. Finally, the number of micro-service replicas that are deployed ranges between 18 and 125, depending on the number of users in the simulation, since more recurrent services are replicated as more users need them to maintain the QoS. The technical details for each of the microservices can be found in Tab. V

Table III: Parameter setting.

Scenario	SDN Nodes	Users	Fog Nodes	Gateways
7-node	7	5	1	1
20-node	20	15	3	2
50-node	50	40	10	5
150-node	150	50	20	10

B. *Response Time-Deployment Cost Trade-off*

The first analysis we propose aims at evaluating the performance of the proposed framework over two different metrics: i) average response time; and ii) deployment cost. At first, each single objective is independently evaluated. Then, both metrics are compared to find a suitable trade-off representing the best deployment cost-response time compromise, giving the same weight to both metrics.

Fig. 3 shows the outcomes of this analysis in terms of deployment cost, whereas Fig. 4 depicts the results in the workflows' average response time. These two figures present the analysis for three groups of simulations: i) the objective function aims at minimizing the average response time; ii) the objective function is defined as the minimization of the deployment cost; and iii) a multi-objective function where the weight given to both metrics is the same. By inspecting Fig. 3, it is clear that the size of the network highly impacts the cost of the associated deployment. However, there are no big differences in the reported cost when the objective of the optimization is varied, i.e., similar results are obtained for the optimization of single metrics as well as for the joint average response time-deployment cost objective. The difference between the joint objective and cost is negligible, while the average response time objectives yields deployments between 467 and 8610€ more expensive. Nonetheless, if we move our attention to the results on Fig. 4, several considerations emerge. At first, the network size negatively impacts the obtained average response time when the optimization objective is the deployment cost, being up to 16.28 times longer in large scenarios compared with smaller ones. Conversely, a

Table IV: Models and costs considered for each infrastructure element.

Element	Model	CAPEX	OPEX
IoT device	Arduino UNO	20 €	$6.10 \cdot 10^{-16}$ €/cycle
IoT device	Texas Instruments CC2538	4.65 €	$1.59 \cdot 10^{-16}$ €/cycle
Fog node	Pandaboard	215 €	$1.48 \cdot 10^{-16}$ €/cycle
Fog node	edge.network instance	0 €	$3.47 \cdot 10^{-16}$ €/cycle
Cloud node	Amazon Web Services m5.xlarge instance	0 €	$5.56 \cdot 10^{-15}$ €/cycle
Cloud node	Google Cloud Platform E2-Standard-4 instance	0 €	$4.17 \cdot 10^{-15}$ €/cycle
SDN switch	Ruijie Networks RG-S5310-24GT4XS	641 €	$1.15 \cdot 10^{-6}$ €/s
SDN controller	Raspberry Pi 3 A+	22.20 €	$2.12 \cdot 10^{-7}$ €/s
Wi-Fi base station	Pulse Electronics TWR0083	21 €	$3.39 \cdot 10^{-8}$ €/s
Bluetooth base station	Pulse Electronics TWR0083	21 €	$1.06 \cdot 10^{-10}$ €/s
ZigBee base station	DIGI XB3-24Z8UM-J	13.73 €	$2.06 \cdot 10^{-8}$ €/s
6LoWPAN base station	Renesas Electronics ZWIR4532-U	28.80 €	$3.82 \cdot 10^{-7}$ €/s

Table V: Input information for micro-services, as reported in [17]

Micro-service	RAM required	Input size	Output size	Execution cycles
ECG and blood pressure monitor	393 MB	8 Kbps	10 Kbps (ECG)/1 Kbps (blood pressure)	$24.44 \cdot 10^9$ cycles
Compression	136 MB	10 Kbps	2.27 Kbps	$9.95 \cdot 10^9$ cycles
Encryption	79 MB	2.27 Kbps (ECG)/1 Kbps (blood pressure)	2.30 Kbps (ECG)/1.02 Kbps (blood pressure)	$6.18 \cdot 10^9$ cycles

longer response time is experienced in smaller scenarios when response time is the unique metric to optimize. The main reason behind this behavior is that smaller scenarios have fewer resources, specially fog nodes, and thus more services need to be deployed to the cloud. Finally, if both metrics are equally balanced, the difference in the experienced response time is negligible.

As a summary, from the previous evaluation , next remarks are extracted. First, the network size is the parameter that impacts the cost of infrastructure deployment, regardless the type of optimization performed. Second, there is a direct proportional relationship between the network size and the average response time when the objective of the optimization is the deployment cost. However, such relationship is inverse when the metric to optimize is the response time. Finally, if a joint optimization is performed, no clear impact is experienced.

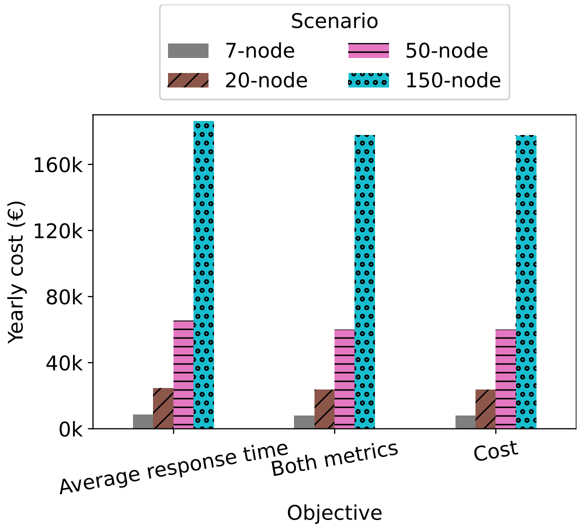


Figure 3: Cost as a function of the optimization objective for the 4 considered scenarios.

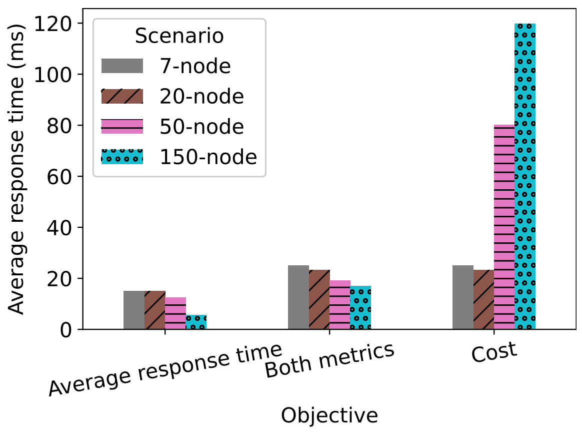


Figure 4: Average response time as a function of the optimization objective for the 4 considered scenarios.

C. Computational Analysis

In the following, an analysis of the computation time required by NIoTO to obtain a solution is performed. NIoTO has been implemented using Python's MIP library [31], solved using Gurobi, and run on a quad-core Intel-based machine at 2 GHz with 16 GB of RAM. Fig. 5 shows the outcomes of the time required to provide an optimal deployment solution as a function of the topology size for the three proposed optimizations.

The first aspect to remark is the exponential increase of the computation time (note the logarithmic y axis) w.r.t. the topology size, for each of the three optimizations. As expected, the joint optimization of response time and cost is the one that requires more time to provide a solution, taking between 2 and 3.5 times more time than the optimization of a single objective. Finally, the metric that in general requires less time to converge is the deployment cost, lasting, on average, 502.55 seconds less than the response time as a single optimization objective. This analysis clearly shows that the MILP version of NIoTO, especially in big scenarios, is mainly suitable for design time optimization. Nonetheless, the development of a faster version of NIoTO, able to optimize and adapt the deployment during execution time, is one of the key future works.

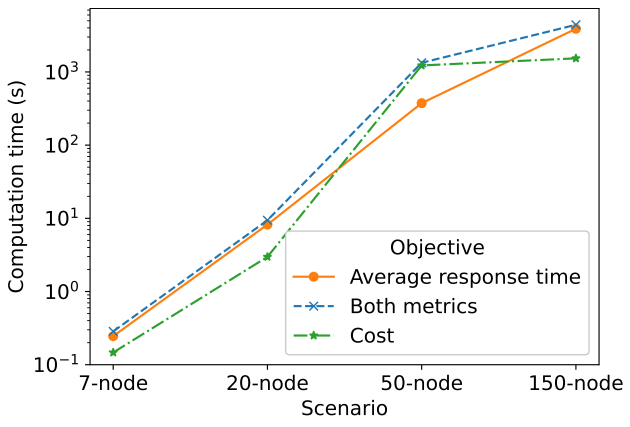


Figure 5: Computation time required to provide an optimal deployment solution as a function of the topology size.

D. Benchmark comparison

The objective of the final analysis is to compare the results obtained by NIoTO with other similar, state-of-the-art techniques. Concretely, two benchmarks are considered for comparison: ModuleMapping [32], as a benchmark aimed at response time and resource usage optimization; and FogPart [33], which focuses on optimizing the financial costs of the deployment. To the best of our knowledge, no benchmarks that jointly optimize response time and cost have been found: ModuleMapping focuses exclusively on response time and ignores cost, while FogPart focuses on cost and ignores response time optimization. Moreover, it is important to note that, out of these three techniques, only NIoTO has a holistic view of the hierarchical multi-dimensional architecture. Nonetheless, the input information was adapted to each of the benchmarks in order to maintain a fair comparison (e.g., FogPart considers the cost of sending data from a computing device to the cloud as a single, aggregated cost, rather than a multitude of CAPEX and OPEX items [33]; and thus, it was provided as an aggregate of the CAPEX and OPEX of the network equipment used to send the data).

The results of the comparison are depicted in Fig. 6 for average response time and in Fig. 7 for deployment cost, while they are detailed in Table VI for response time and Table VII for cost. Moreover, the comparison includes two categories for NIoTO (one for the appropriate objective and one for the joint approach), and a single one for the benchmark that optimizes the appropriate objective (i.e., ModuleMapping for response time and FogPart for cost). Starting the comparison with response times (Fig. 6, Table VI), we find NIoTO with the average response time objective as the best technique for all the analyzed scenarios, which we consider to be optimal. Its performance is closely followed by NIoTO optimizing both metrics, with an average optimality gap of approximately 9.16 ms. ModuleMapping is the last one, with an average optimality gap of 40.22 ms. The response time of ModuleMapping is constant through all the topology sizes due to the fact it only considers two of the three dimensions: application and computing. ModuleMapping uses the computational resources of a device, as well as the resources required by the micro-service, as the main metric to select where to deploy a micro-service to [32]. Since the cloud is consistently the most powerful and resourceful computing device, ModuleMapping will deploy as many micro-services as possible there. Due to the usage of fog nodes to reduce the experienced latency, NIoTO achieves an average speed-up of 5.11 w.r.t. ModuleMapping.

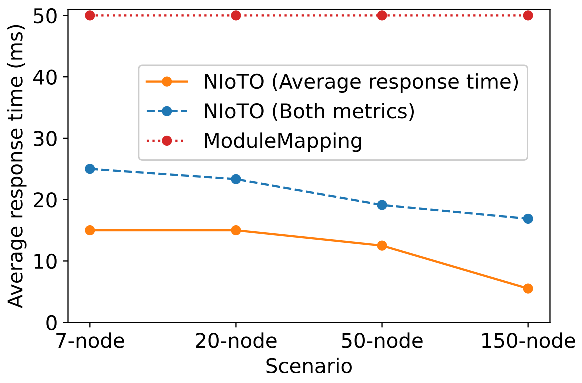


Figure 6: Average response time comparison of NIoTO and ModuleMapping.

Table VI: Average response times yielded by NIoTO and ModuleMapping.

Scenario	NIoTO (average response time)	NIoTO (both metrics)	ModuleMapping
7-node	15.005 ms	25.004 ms	50.004 ms
20-node	15.005 ms	23.338 ms	50.004 ms
50-node	12.506 ms	19.120 ms	50.004 ms
150-node	5.508 ms	16.882 ms	50.004 ms

Continuing with cost, as depicted in Fig. 7 and detailed in Table VII, NIoTO with the cost objective is the optimal solution for the analyzed scenarios. Once again, we find the next best solution to be NIoTO optimizing both metrics, with an optimality gap of approximately 50€ (0.08%). FogPart is the third best technique, with an optimality gap of 3954€ (6.22%). It is important to note that the gap between FogPart and NIoTO increases with topology size, and thus, using FogPart in larger topologies may lead to larger optimality gaps. The difference in costs responds to FogPart's partial view on the scenario, not considering the computing dimension. To choose the node to deploy a micro-service to, FogPart compares the cost of communicating the previous micro-service in the workflow with the cloud, and the cost of communicating with a fog node instead, choosing the most cost-effective communication [33]. Thus, the costs of the computing dimension, such as the CAPEX of the computing devices used or their OPEX per cycle, may not be optimal. This is precisely the difference we find between NIoTO and FogPart. In a general conclusion, we find NIoTO, with the according objective, as the optimal solution in terms of response time and cost. Moreover, the version of NIoTO that optimizes both metrics at the same time is able to perform better at both dimensions than state-of-the-art techniques aimed at their optimization.

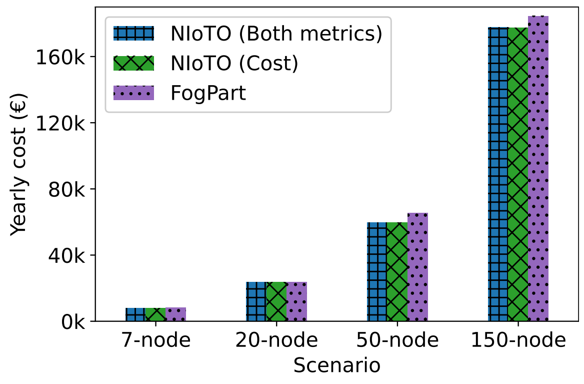


Figure 7: Deployment cost comparison of NIoTO and FogPart.

Table VII: Yearly costs of deploying the scenario using NIoTO and FogPart.

Scenario	NIoTO (cost)	NIoTO (both metrics)	FogPart
7-node	8,070€	8,070€	8,233€
20-node	23,672€	23,673€	23,673€
50-node	59,801€	59,801€	65,644€
150-node	177,508€	177,772€	184,502€

VI. CONCLUSIONS AND FUTURE WORK

As the number of IoT devices grows every year, the demand for QoS-strict IoT applications, hardly suitable for a cloud-based deployment, does as well. A hierarchical multi-dimensional architecture is an enabler for this kind of applications, but optimizing the QoS requires the consideration of the interplay between the computing, networking and application dimensions. In this work, we present NIoTO, a framework to optimally deploy next-gen IoT applications in a hierarchical multi-dimensional architecture, considering all the three aforementioned dimensions. NIoTO is able to optimally assess the number of micro-service replicas in the application dimension, their deployment in the computing dimension, and the placement of the SDN controller in the networking dimension, along with optimizing the routing of the generated traffic flows. The joint consideration of all three dimensions, and the optimization of the response time and cost, including the assessment of the optimal trade-off between them, allows it to optimize each of the metrics further than related, state-of-the-art frameworks. In the future, we expect to support run-time, dynamic optimizations of the deployment through the development of a faster solver for NIoTO, enabling it to adapt the deployment to environmental changes. Moreover, we expect to evaluate NIoTO's performance over real or emulated network test-beds.

References

[1] Cisco. (2020, March) Cisco Annual Internet Report (2018–2023). (visited on Jan. 29, 2021). [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf

[2] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini, and A. Zanni, "A survey on fog computing for the internet of things," *Pervasive and Mobile Computing*, vol. 52, pp. 71 – 99, 2019.

[3] B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant, and K. Mankodiya, "Towards fog-driven iot ehealth: Promises and challenges of iot in medicine and healthcare," *Future Generation Computer Systems*, vol. 78, pp. 659–676, 2018.

[4] A. Brogi, S. Forti, C. Guerrero, and I. Lera, "How to place your apps in the fog: State of the art and open challenges," *Software: Practice and Experience*, vol. 50, no. 5, pp. 719–740, 2020.

[5] S. Forti and A. Brogi, "Continuous reasoning for managing next-gen distributed applications," *arXiv preprint arXiv:2009.10245*, 2020.

[6] L. Nkenyereye, J. Y. Hwang, Q.-V. Pham, and J. S. Song, "Virtual iot service slice functions for multi-access edge computing platform," *IEEE Internet of Things Journal*, 2021.

[7] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in sdn," *IEEE Communications Surveys & Tutorials*, pp. 472–503, 2019.

[8] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[9] D. Kutscher, T. Karkkainen, and J. Ott, "Directions for Computing in the Network," Internet Engineering Task Force, Internet-Draft draft-kutscher-coinrg-dir-02, Jul. 2020, (visited on Jan. 29, 2021). [Online]. Available: https://datatracker.ietf.org/doc/html/draft-kutscher-coinrg-dir-02

[10] J. L. Herrera, J. Galán-Jiménez, J. Berrocal, and J. M. Murillo, "Optimizing the response time in sdn-fog environments for time-strict iot applications," *IEEE Internet of Things Journal*, 2021.

[11] K. Indrasiri. Microservices in practice - key architectural concepts of an MSA. (visited on Jan. 14, 2021). [Online]. Available: https://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/

[12] H. Haas, D. Orchard, F. McCabe, C. Ferris, E. Newcomer, D. Booth, and M. Champion, "Web services architecture," W3C, W3C Note, Feb. 2004, https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[13] GRPC, "Documentation — gRPC," 2021. [Online]. Available: https://grpc.io/docs/

[14] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[15] R. Priya, K. H. Prabha, R. Hemalatha, P. Vanmathi, and M. Ilakiya, "A secured remote health monitoring system based on iot," *Annals of the Romanian Society for Cell Biology*, pp. 17 759–17 765, 2021.

[16] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare internet of things: A case study on ecg feature extraction," in *2015 IEEE international conference on computer and information technology; ubiquitous computing and communications; dependable, autonomic and secure computing; pervasive intelligence and computing*. IEEE, 2015, pp. 356–363.

[17] A. Limaye and T. Adegbija, "A workload characterization for the internet of medical things (iomt)," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 302–307.

[18] M. Huang, W. Liu, T. Wang, H. Song, X. Li, and A. Liu, "A queuing delay utilization scheme for on-path service aggregation in services-oriented computing networks," *IEEE Access*, vol. 7, pp. 23 816–23 833, 2019.

[19] Q. Duan, S. Wang, and N. Ansari, "Convergence of networking and cloud/edge computing: Status, challenges, and opportunities," *IEEE Network*, vol. 34, no. 6, pp. 148–155, 2020.

[20] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017.

[21] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, and Y. Sun, "Minimizing controller response time through flow redirecting in sdns," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 562–575, 2018.

[22] ETSI. Open Source NFV Management and Orchestration (MANO). (visited on Jan. 14, 2021). [Online]. Available: https://www.etsi.org/technologies/open-source-mano

[23] M. M. Badawy, Z. H. Ali, and H. A. Ali, "Qos provisioning framework for service-oriented internet of things (iot)," *Cluster Computing*, pp. 1–17, 2019.

[24] J. Berrocal, J. Garcia-Alonso, C. Vicente-Chicote, J. Hernández, T. Mikkonen, C. Canal, and J. M. Murillo, "Early analysis of resource consumption patterns in mobile applications," *Pervasive and Mobile Computing*, vol. 35, pp. 32 – 50, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119216300797

[25] M. Ronchetti, G. Succi, W. Pedrycz, and B. Russo, "Early estimation of software size in object-oriented environments a case study in a cmm level 3 software firm," *Information Sciences*, vol. 176, no. 5, pp. 475–489, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025505000083

[26] M. Bichier and K.-J. Lin, "Service-oriented computing," *Computer*, vol. 39, no. 3, pp. 99–101, 2006.

[27] Open Networking Foundation, "OpenFlow Switch Specification 1.3.0," pp. 1–3205, 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

[28] P. Erdös and A. Rényi, "On random graphs i," *Publ. math. debrecen*, vol. 6, no. 290-297, p. 18, 1959.

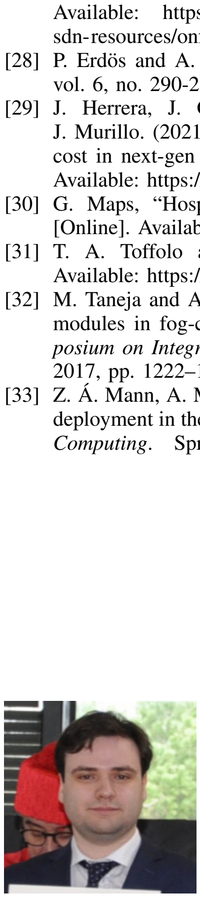
[29] J. Herrera, J. Galán-Jiménez, J. García-Alonso, J. Berrocal, and J. Murillo. (2021) Joint optimization of response time and deployment cost in next-gen iot applications. (visited on Oct. 14, 2021). [Online]. Available: https://tinyurl.com/nioto-capex-opex

[30] G. Maps, "Hospital provincial nuestra sra. de la montaña," 2021. [Online]. Available: https://goo.gl/maps/NqsPvQbw7JhaSW8k6

[31] T. A. Toffolo and H. G. Santos, "Python-MIP," 2021. [Online]. Available: https://www.python-mip.com/

[32] M. Taneja and A. Davy, "Resource aware placement of iot application modules in fog-cloud computing paradigm," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 1222–1228.

[33] Z. Á. Mann, A. Metzger, J. Prade, and R. Seidl, "Optimized application deployment in the fog," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 283–298.



Juan Luis Herrera received Bachelor's degree in software engineering from the University of Extremadura in 2019. He is a researcher in the University of Extremadura's Computer Science and Communications Engineering Department. His main research interests include IoT, fog computing and SDN.

Jaime Galán-Jiménez received the Ph.D. in computer science and communications from the University of Extremadura in 2014. He is currently with the Computer Science and Communications Engineering Department, University of Extremadura, as an Assistant Professor. His main research interests are Software-Defined Networks, 5G network planning and design, and mobile ad-hoc networks.

Jose Garcia-Alonso (IEEE Memeber) is an Associate Professor at the University of Extremadura. His research interests include software engineering, mobile computing, pervasive computing, eHealth, gerontechnology.

Javier Berrocal (IEEE Memeber) is a cofounder of Gloin. His main research interests are software architectures, mobile computing and edge and fog computing. Berrocal has a PhD in computer science from the University of Extremadura, where he is currently an associate professor.

Juan M. Murillo (IEEE Member) is a cofounder of Gloin and a full professor at the University of Extremadura. His research interests include software architectures, mobile computing, and cloud computing.